UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

ACTIVISION BLIZZARD, INC.,
ELECTRONIC ARTS INC.,
TAKE-TWO INTERACTIVE SOFTWARE, INC.,
2K SPORTS, INC., and
ROCKSTAR GAMES, INC.,
Petitioner

v.

ACCELERATION BAY, LLC,
Patent Owner

Case IPR2016-00747
Patent 6,732,147

DECLARATION OF MR. VIRGIL BOURASSA IN SUPPORT OF PATENT OWNER'S RESPONSE

I, Virgil E. Bourassa, declare as follows:

1. I am over the age of majority and make this declaration of my own personal knowledge.

2. Between October, 1996 and May, 2001, I worked in Boeing's Mathematics and Computing Technology research division as a computing technologist. As a computing technologist, I spent my time augmenting [redacted]

3. [redacted]. Boeing used [redacted] in order to [redacted]. In November 1996, Robert Abarbanel, the head of [redacted] Research, asked me to start developing a communications library for [redacted]. We referred to the communications library as SWAN, which resulted in inventions covered by U.S. Patent Nos. 6,829,634 ("the '634 Patent"), 6,701,344 ("the '344 Patent"), 6,920,497 ("the '497 Patent"), 6,910,069 ("the '069 Patent"), 6,732,147 ("the '147 Patent"), and 6,714,966 ("the '966 Patent") (collectively, the "SWAN Patents"). I am a co-inventor of the SWAN Patents.

4. At around this time, Boeing was expanding beyond the Puget Sound region as it acquired a series of airplane companies. For these reasons, we wanted to augment [redacted] to allow for peer-to-peer communications for virtual communications so that review sessions could be shared across the world, to allow major portions, such as the database query used for selecting parts, to be broken

out, and to allow new modules to be introduced without a major software overhaul. Boeing had a procedure that allowed for two running [redacted] processes to work in tandem. However, as soon as a third [redacted] process was introduced, it broke down.

5. As Fred and I described in our Invention Disclosure Form, the typical computer network communications techniques included: (1) fully-connected point-to-point network protocols, (2) client/server middleware, (3) multicasting network protocols, and (4) peer-to-peer middleware.

6. Fully-connected point-to-point networking protocols were disadvantageous because fully-connected network graphs do not scale as the number of participating processes grows. Client/server middleware systems were disadvantageous because the server is a performance bottleneck and a single point of failure. Multicast networking protocols were disadvantageous because multicast traffic at the time was limited to single local-area networks. Peer-to-peer middleware at the time was not suitable for the needs of medium to large-scale collaboration.

7. The lack of technology available that could provide peer-to-peer communications among computer processes across the world with high reliability and low latency reaffirmed that a solution was needed in this area to satisfy an existing need in the market. Because I had written a reliable UDP multicast

communications library suitable for local-area networks for another project unrelated to [redacted], Robert Abarbanel, my supervisor and the head of [redacted] research at the time, asked me to come up with a way to create something similar for the wide-area networking required to "[redacted]" across multiple [redacted] sessions across the country.

8. I originally anticipated this to be a simple task and expected that I would have a working model within three weeks. This estimate was far off from what I was expecting—instead, it took me about three years and about 28 epiphanies to incorporate into [redacted] what would eventually be referred to as SWAN.

9. I knew that I did not want to use a tree per se because that would put the root node in a privileged position. I considered breaking the nodes into two trees that would be connected between the pair of roots and amongst the leaf nodes. In around December of 1996, I began working with Fred Holt, a colleague at Boeing with a Ph.D. in Mathematics, and told him about my proposal. The idea worked well for a pair of ternary trees up to a depth of three. However, Fred determined that beyond this level, there was insufficient connectivity among the leaves to share the information between the trees any more quickly than sending the shared message to the other through the roots. In other words, at this size and

above, some of the leaf nodes would take at least twice as long to share a message as the root nodes.

10. Fred and I first tested a 3-regular (meaning 3 connections per node) and 3-connected (meaning at least 3 nodes would have to be lost to break the graph apart) graph, but it only worked well while the number of nodes in the graph was even. If the number of nodes were odd, it would create a half-edge problem, meaning that it was necessary to keep track of one special node. Fred and I discovered that if we made a four-regular, four-connected graph, the half edge problem in the 3-regular, 3-connected graph was eliminated.

11. We then realized we could solve the half-edge issue by creating a 4-regular, 4-connected graph that had a low diameter. The diameter was our figure of merit for the latency of the graph. We gravitated towards developing an inductive algorithm to build our 4-regular, 4-connected graphs. We would have a known node in the graph to be the point of entry, then using an inductive approach, have new nodes added in the vicinity of this entry point in such a way as to maintain 4-regularity and connectedness. However, when we tested at 20 nodes, we had a diameter of four, which seemed high to me. To confirm this, I ramped up the algorithm to build a graph of 2000 nodes, and got a diameter of 179. Since a balanced binary tree with 2000 nodes would only have a diameter of 20, we clearly had done something wrong.

12. We then realized that we needed to create 4-regular, 4-connected graphs with as low as a diameter as we could get. This was when we decided to use genetic algorithms to evolve lower and lower diameter graphs. We started with randomly generated 4-regular, 4-connected graphs, and then had each "generation" mate those with the lowest diameters to achieve even lower diameter graphs. Fred and I also realized that we wanted to build the graph in such a way as to require only local knowledge, not global knowledge. Normally with genetic algorithm optimization, the initial random population individuals leave much to be desired because it takes dozens or hundreds of generations to evolve "fit" individuals. However, in this exercise, a first generation single individual of 2000 nodes had a diameter of only 9. This allowed us to realize that introducing nodes into the graph "damaged" the pathways they were injected on, making some paths between a pair of nodes longer. Therefore, while the inductive algorithm concentrated damage in the area of the contact node, random injection spread the damage around evenly.

13. At this point, we were closer to reaching a solution. We wanted to weave together a "fabric" of point-to-point connections, such that each process was connected to exactly four others. For reliability of the overall session, we wanted the graph to be four-connected, meaning that at least four nodes would have to fail

to separate the graph into two pieces. We also wanted to build the graph randomly in such a way as to require no global knowledge, only local knowledge.

14. I built a two-dimensional visualization tool to watch the graphs as they were formed and to measure their resulting connectivity and diameter. This visualization tool helped us determine how to realize our invention beyond the theoretical idea.

15. I wanted to make sure that once we had our basic 4-regular, 4-connected graph that nodes would be able to be added and maintained. I discovered that if we took two randomly selected disjointed edges, and replaced those edges with edges to the new node, we would be able to meet this goal. As indicated in my Invention Disclosure Form, I referred to this as "clothes-pinning." *See* Ex. 2008, Fig. 5:

16. This way, the new node will have four neighbors, and the previous nodes all maintain four neighbors as well. The pathways along these two edges are "damaged" with the addition of the new node. All paths between the nodes along

these two edges are now one node further apart. Thus, choosing them randomly spreads the damage around.

17. We were concerned whether clothes pinning ran the risk of reducing the graph's reliability. Fred evaluated all of the cut-set combinations. He determined that given a cut-set where the two edges chosen are the only ones coming from a potential partition, a clothespin operation would reduce the cut set from four nodes to three. However, upon further investigation, a node with only one connection to a partition can only arise from a previous clothes pinning on a node with only one connection to the partition to begin with. In other words, to get to the dangerous state, you had to start in a dangerous state. Since the graph is built incrementally from a completely connected, four-regular, four-connected graph of five nodes, there is no dangerous state so there is no way to enter a dangerous state by the process of clothes pinning new nodes into the graph incrementally. Fred proved to me by induction that clothes pinning maintains four-connectedness, in addition to four-regularity and low latency.

18. We also wanted to know the full state of the graph, put all the edges into a hat, and pull out two at random and confirm that they do not share a node between them. In order to do that, we needed global knowledge and that the graph not be in the process of changing. Both of these requirements went against our design tenets since we only had local knowledge. A new participant wanting to

join the graph could contact any existing node in the graph and request inclusion. We could approximate a random edge selection by requesting an edge after a "drunk walk." To do this, the node contacted chooses one of his existing neighbors randomly, and forwards the request to that node. It, in turn, does the same thing, and the random walk continues until finally the last edge is selected.

19. We experimented with a 100-node 4-regular, 4-connected random graph, with a diameter of 6. Using Einstein's theory of Brownian Motion, we confirmed that at a drunk-walk of length 36, the diameter squared, the probability of landing on any given node was nearly identical. A walk of length 6 led to a terrible distribution, mostly centered on the entrance node and its neighbors. A walk twice that long was better, with about half of the nodes having some preference over the other half. Since twice the diameter was almost as good as the diameter in algorithmic order, and since the slightly preferred half switched by adding one to this length, we opted for getting our two "random" edges by taking the first drunk walk at twice the diameter, and the second at twice the diameter plus one more, which we referred to as a "staggered" drunk walk. The result was an efficient selection algorithm with results practically identical to true random selection.

20. For a message to be shared in the SWAN graph, a source process creates a message and then repeats it to each of its four neighbors. These in turn,

repeat it to each of their three other neighbors, ignoring the message if they see it a second time. By including a count of how many times a message has been forwarded, every node in the graph can see how far away they are from the source code of the message—it is the least distant copy of that message they receive.

21. The diameter is determined in a distributed fashion. In the small regime (less than five nodes, kept fully connected), each node is initialized to understand that the diameter of the graph is one hop. As nodes are added and the graph enters the large regime (four-regular), at some point the diameter increases. When the diameter increases, either of the two nodes furthest away from each other will recognize it upon hearing a message from the other. That node will then know that the diameter has increased. When a node discovers a new diameter, it sends a special administrative message through the graph to let everyone else know.

22. There are redundant pathways, which provide reliability in the face of node and link loss during the sharing of messages. These also provide faster alternative pathways in the face of slow links.

23. Another problem to be tackled was making guarantees to the consuming applications about the nature of the messages being shared. We provided that the messages from multiple sources may come in any order, but messages from the same source would be delivered in the order they were sent.

And when a new node joins the graph, it would not get what happened in the past, but subsequent message streams from each source would be intact. Assigning each message a source, and an index specific to that source, then queuing up messages from each source that arrived out-of-order until missing messages arrive, solved this need once communication was underway.

24. But for newcomers, there was a potential problem. Because of the distributed nature of the graph, there was no guarantee that these messages will arrive in order. A new node, which I refer to as a greenhorn, may have been injected into the graph after, say, message 143 from Oz has gone by, but a slower delivery of message 142 from Oz shows up. The greenhorn's Swan library callback would deliver the Oz:142 message to its application, then queue up messages from Oz forever after waiting for Oz:143 to show up.

25. I concluded that the greenhorn could not solve this problem. Instead, it was up to its neighbors, which know that the newcomer is new, because they just made connections to it. When this happens, they put the newcomer into a "greenhorn" status.

26. While in this status, they don't just send him copies of messages sent from their other neighbors. Instead, they treat him the same as the application, only sending messages from each source in order. In this way, they remove any holes in the message streams from each of the newcomer's neighbors individually.

27. Once a neighbor has no messages backed up in its queues, i.e., all of the “holes” have been removed, they can stop treating the newcomer as a greenhorn, and go on to feeding him the messages as they are received. In this way the newcomer never encounters a hole in its incoming streams.

28. When a node is ready to leave the graph, it first provides all of its neighbors with a list of all of its other neighbors. The first pair of neighbors then disconnect and pair up with each other, as does the second pair.

29. If, however, a neighbor dies unexpectedly, its neighbors discover this the next time they try to send it a message. Upon discovering the lost neighbor, each will send an administrative message through the fabric asking for anyone else needing a connection to contact it.

30. These methods of repair work well the vast majority of the time. However, there is a small possibility (at most one in nine, but diminishing as the size of the graph grows) of a problem.

31. Consider a list of neighbors in which the first pair is already connected to one another. Now, this pair of nodes cannot connect to each other and maintain 4-connectedness. We refer to this situation as a “bilateral 3-connected wedgie.” The solution we came up with was to move the problem somewhere else. When node A needs a neighbor, it sends an “I need a connection” message through the fabric, but includes its list of neighbors in the message.

32. Now when B sees the message, realizes that it too needs a connection, but is already connected to A. First, it checks to see if it has the same set of neighbors as A. If so, the graph has actually just gone from 5 nodes total down to 4, which looks like a wedgie but is just a transition from the large regime to the small regime.

33. If it has different neighbors, it's a wedgie, and B knows it. B invokes the "wedgie repair algorithm" as follows. B contacts a random neighbor, C, instructing it to disconnect from one of its other neighbors, D, again using a random choice. C immediately replaces this disconnected neighbor with a connection to A.

34. This results in the missing connection moving from A, which wasn't working as a potential connection for B, to some other node, D, which has a 90%+ chance of being suitable. Now D sends an "I need a connection" message through the fabric, B picks it up, and connects to D. If C had happened to choose a random neighbor that was also connected to B, the process would just repeat until the bad luck was resolved.

35. Although SWAN was built by August 30, 1999, we were finally able to run it within [redacted] without any software bugs by September 16, 1999, as indicated on the Invention Disclosure Form. In fact, Fred and I presented a

demonstration of SWAN to the [redacted] Group prior to September 16, 1999.

36. SWAN took us 3 years to complete and was fully built as of August 30, 1999, and successfully implemented in [redacted] by September 16, 1999, as indicated on our Invention Disclosure Form, attached to the Patent Owner Response as Ex. 2008. The Invention Disclosure Form contains my signature on each page following the first page. Although my signature is dated December 22, 1999, the Invention Disclosure Form describes how SWAN was working in [redacted] as of September 16, 1999. This is indicated on the first page of the document, which states that the SWAN was satisfactorily tested on September 16, 1999. In addition to [redacted], SWAN had undergone alpha and beta testing for use in Boeing's [redacted] used in Everett, Washington.

37. The Invention Disclosure Form describes the SWAN technology, which was implemented as a 4-regular network. By September 16, 1999, SWAN used an incomplete graph. Each participant had a connection to at least four neighbor participants. In operation, SWAN would send data from an originating participant to the other participants by sending data through each of its connections to its neighbor participants. Each participant would then send data that it receives from a neighbor participant to its other neighbor participants. SWAN was also a dynamic network that used a non-routing table based broadcast channel. SWAN

was an overlay network that used an underlying communication network. SWAN provides peer-to-peer communications between the participants connected to the broadcast channel. In one example, each participant to the broadcast channel could receive an indication of the four neighbor participants. The broadcast component could receive data from a neighbor participant using the communication network and send the data to the neighbor participants to affect the broadcasting of the data to each participant of the broadcast channel.

38. SWAN was able to accommodate any number of nodes being brought in or removed in any order without disruption to the session conversation. In other words, since the system was completely distributed among the participants, any of them could join, depart, or fail at any time and in any order, without causing any interruptions to the connections.

39. To summarize, the process that resulted in the creation of the SWAN Patents took Fred Holt and I almost three years, although I estimated to Robert that I should be able to have something working in about three weeks. However, Fred Holt and I successfully implemented SWAN as described in our Invention Disclosure Statement in [redacted] by September 16, 1999. SWAN was able to provide general world-wide peer-to-peer communications among computer processes.

40. In addition, Boeing launched its [redacted] initiative to look for Boeing-internal technologies that had commercial potential. SWAN was one of the companies selected and quickly rose to become the leader in this portfolio of potential Boeing spinout companies.

41. In connection with writing this declaration, I reviewed the Invention Disclosure Form, which is attached to the Patent Owner Response as Exhibit 2008. Fred Holt and I completed the Invention Disclosure Form after we created SWAN. It was the regular practice at Boeing to complete and maintain such invention disclosure forms. Because it was typical practice for our team to include dates on documents, any dates that are indicated on the Invention Disclosure Form are reliable evidence of when the described events occurred. Specifically, SWAN was built on August 30, 1999, and satisfactorily tested by September 16, 1999. Exhibit 2008 has not been altered from what was provided to counsel, other than the following markings at the bottom of the page: "Protective Order Material," "Patent Owner Acceleration Bay, LLC," the exhibit number, and page number. No other changes have been made to this document.

42. I have kept Exhibit 2008 in my possession since the document was created at Boeing until it was transferred to Panthesis as part of the spin-off of the company. After Panthesis ceased its operations, I kept this document in my personal possession.

43. I declare under penalty of perjury under the laws of the United States of America that this declaration is true, complete, and accurate to the best of my knowledge. I further acknowledge that willful false statements and the like are punishable by fine or imprisonment, or both under 18 U.S.C. § 1001.

Executed at Bellevue, Washington on December 12, 2016.

Virgil E. Bourassa